

AD-A169 615

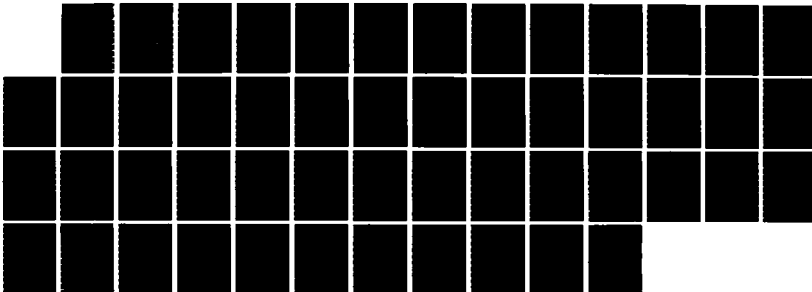
PSYCHOLOGICAL CONCEPTS IN A PARALLEL SYSTEM(U) BROWN
UNIV PROVIDENCE RI CENTER FOR NEURAL SCIENCE
J A ANDERSON ET AL. 27 JUN 86 N00014-81-K-0136

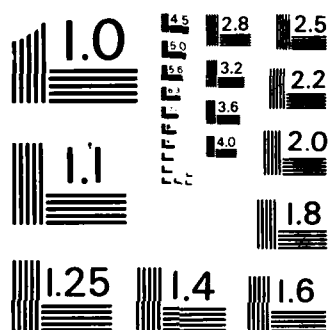
1/1

UNCLASSIFIED

F/G 5/10

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

12

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER #30	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Psychological Concepts in a Parallel System		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James A. Anderson and Gregory L. Murphy		8. CONTRACT OR GRANT NUMBER(s) N000-14-K-0136
PERFORMING ORGANIZATION NAME AND ADDRESS Center for Neural Science Brown University Providence, Rhode Island 02912		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N-201-484
CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training Research Program Office of Naval Research, Code 442PT Arlington, Virginia 22217		12. REPORT DATE June 27, 1986
MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES 48 pages
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited. Publication in part or in whole is permitted for any purpose of the United States Government.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

To be published in Evolution, Games and Learning, ed. by J. D. Farmer, A. Lapides, N. Packard, B. Wendroff; North Holland.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Concepts
- Prototypes
Neural Models
Neural Nets. C

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Many parallel information processing systems have been proposed which are loosely based on the architecture of the nervous system. Many of the testable predictions of these systems fall in the realm of cognitive science since they perform some 'computations' well and some poorly and have pronounced 'psychologies'. In such system, simple elements are connected to each other; the elements act like 'neurons' in that they sum excitation and inhibition from other elements. Information is represented as large state

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

86 7 3 010

vectors of element activities, and it can be shown that simple 'synaptic' learning rules can serve to associate arbitrary state vectors using a matrix of connections. One version of this class of models has been shown, in the past to form concepts and to perform classification computations. This paper discusses the general importance and structure of psychological concepts and describes three simulations of a simple neural model that attempts to reproduce a few of the important properties of human concept formation.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A-1	



Psychological Concepts in a Parallel System

James A. Anderson

and

Gregory L. Murphy

Department of Psychology,
and Center for Cognitive Science
Brown University
Providence, RI 02912

Running Title: Concepts in a Parallel System

*This work was supported in part by a grant from the National Science Foundation, Memory and Cognitive Processes Section (Grants BNS-82-14728 and BNS-83-15145) and by the United States Office of Naval Research under Contract N00014-81-K-0136.

Abstract

Many parallel information processing systems have been proposed which are loosely based on the architecture of the nervous system. Many of the testable predictions of these systems fall in the realm of cognitive science since they perform some 'computations' well and some poorly and have pronounced 'psychologies'. In such system, simple elements are connected to each other; the elements act like 'neurons' in that they sum excitation and inhibition from other elements. Information is represented as large state vectors of element activities, and it can be shown that simple 'synaptic' learning rules can serve to associate arbitrary state vectors using a matrix of connections. One version of this class of models has been shown in the past to form concepts and to perform classification computations. This paper discusses the general importance and structure of psychological concepts and describes three simulations of a simple neural model that attempts to reproduce a few of the important properties of human concept formation.

Psychological Concepts in a Parallel System

Parallel systems composed of very many interconnected elements are both simple brain models and possible novel computer architectures. Potential advantages of such systems, as is well known, are massive parallelism with resulting speedup of computation as well as general ability to compute with noisy, corrupted, or missing data. Parallel, distributed, associative models have pronounced psychologies. Some ways of handling information are natural for them, and some things that we might want them to do are unnatural and quite difficult to do easily. Since these models virtually all had their beginnings as 'brain' models, a question of considerable interest is whether their capabilities are features of human psychology. We know a great deal about some of the ways humans combine and use information, often in considerable quantitative and qualitative detail.

Human Concepts. Many of the systems discussed at this conference have been designed to perform some sort of intelligent behavior. However, few of them have been specifically designed to simulate human behavior. In this chapter, we will present a parallel memory model of neuron-like elements that shows many of the behaviors characteristic of the human conceptual system. People form concepts that correspond to categories of objects in their world, and this is one of the most important methods of knowledge representation that they have. Simulating human concepts in a dynamic memory model would be a significant advance in understanding how people form new concepts and would also demonstrate the usefulness of such models in generating intelligent behavior.

It takes only a moment's thought about concepts to understand why someone would say that the topic of concept formation is "about the most fundamental problem of cognitive psychology" (Jackendoff, 1983, p. 87). In fact, without concepts, memory would be nearly useless: although we might remember specific experiences, we would have no means of generalizing our experiences to new situations. For example, it is our concept of robin that tells us that a new object we see is a harmless animal that will chirp and eat worms. Our concept of tiger tells us not to stick our hands through the bars at the zoo--even though we've never seen this particular animal before. Smith and Medin (1981) go so far as to say that "without concepts, mental life would be chaotic" (p. 1).

Concepts, then, are of great importance to many intelligent behaviors: memory, pattern recognition, problem solving and understanding among them. It is not surprising that they have become a central issue of cognitive psychology. Psychologists investigate how people learn concepts, how they are represented in the mind, how they are used to categorize the world, and how they are used to answer questions and solve problems. Although the human conceptual system is by no means the only way to represent knowledge, any system that captures its properties will be well on the way to intelligent behavior. The remainder of this chapter will be devoted to illustrating how a parallel memory model can capture some of the main properties of the human conceptual system.

At a general level, it is easy to explain what a concept is: a concept is a mental representation or rule that picks out a class of objects or events. However, this description may lead to an oversimplified notion of what concepts are like. Specifically, one might have the idea that a concept is like a definition, that it gives the criteria by which all and only the members of the concepts are identified. However, one important fact about human concepts is that they are fuzzy. They do not provide all-or-none criteria for defining classes of objects: in many cases, there probably are no such criteria, and when the criteria do exist, people often do not know what they are.

The fuzziness of concepts has two specific components. First, it includes the phenomenon of unclear examples--objects which do not fall clearly into or out of some category. For example, a tomato is not clearly either a fruit or a vegetable. Most people have experienced socially embarrassing unclear cases such as ashtray-bowls and there are more puzzling cases in the sciences, in which arguments about the correct categorization of an unclear example may carry on for years. In everyday life, we often encounter unclear cases simply because we have limited information about the object or event. But if our concepts were like definitions, every object would be clearly in or out of a concept. These unclear cases do not demonstrate a fault of our concepts but rather the unavoidable variation and uncertainty in the world.

The second component of fuzziness is graded category membership, often called typicality. Even among objects that can definitely be categorized, some objects seem to be "better"

category members than others: a robin is a better bird than a chicken is; a trout is a better fish than an eel is. People's judgments of how typical objects are of some concept are highly predictable, and typical objects can be classified more quickly and with greater reliability than atypical objects (Rips, Shoben & Smith, 1973; Rosch, 1975; Rosch & Mervis, 1975).

One way to explain concept fuzziness is to propose that people learn a prototype -- the most typical example -- in order to learn the concept. People's prototype of bird is a small, flying animal with wings that eats worms and builds a nest. Fuzziness results from the continuous gradation of similarity of objects to the prototype. For example, most robins are similar to this prototype but most penguins are dissimilar to it, explaining why people are faster at identifying robins as birds. Rosch (1978) and Smith and Medin (1981) provide good reviews of such theories. However, many models of concepts in cognitive science seem to ignore both unclear examples and typicality structure (see Cohen & Murphy, 1984).

Another important aspect of human concepts is their hierarchical structure. Concepts seem to be organized in a nested fashion: robin-bird-animal-living thing or rocking chair-chair-furniture-artifact, for example. This structure allows people to economically represent information about increasingly general categories (Collins & Quillian, 1969). So, the facts people know about birds apply to robins, larks, crows, etc., and the facts people know about animals apply to birds, mammals, reptiles, etc. These hierarchies allow us to draw inferences about objects. For example, if Tweety is a bird, we

can deduce that Tweety is an animal. Such inferences are important, because we can now infer properties true of all animals to be true of Tweety as well. This system is economical in that information about all animals is not repeated with each animal, but is stored as true about the animal concept in general. Exactly how people represent such hierarchies is still a matter of contention in psychology (see Smith, 1978), but it is clear that these hierarchies are important means of organizing concepts and storing information.

This review should suggest the challenge faced by any model of concepts. It should be able to operate in an ambiguous world, which may contain objects that do not fit into any category clearly. It should reflect the typicality structure of concepts: some members should be 'better' examples of the concept than others. The model should also represent the hierarchical relations among concepts; it should be able to make inferences about the attributes of objects, given their category membership. The next section outlines a parallel memory model and demonstrates how these goals can begin to be met.

Stimulus Coding and Representation. We have billions of neurons in our cerebral cortex. The cortex is a layered two dimensional system which is divided up into a moderate number of subregions. The subregions project to other subregions over pathways which are physically parallel. Much, perhaps most, of the computational power of the brain is in the details of the neural codes, i.e. the biologically determined representation of the stimulus. Possibly the brain is not very smart. It does little clever computation but powerful, brute force operations on

information that has been so highly processed that little needs to be done to it. However the pre-processing is so good, and the numbers of elements so large that the system becomes very powerful as a result.

Our fundamental modelling assumption is that information is carried by the pattern or set of activities of many neurons in a group of neurons. This set of activities carries the meaning of whatever the nervous system is doing. We represent these sets of activities as state vectors. Percepts, or mental activity of any kind, are similar if their state vectors are similar. Our basic approach is to consider the state vectors as the primitive entities and see how state vectors can lawfully interact, grow and decay. The component values in the state vectors might correspond to the activities of moderately selective neurons. Other possible neurobiological candidates for elements are cortical columns or small groups of neurons. The computational behavior of the models is indifferent to the exact anatomical attribution of the elements, though they generally seem more analogous to single neurons than cortical columns, which may contain tens of thousands of neurons. Many working in the field prefer to call the basic units, the components of the state vectors, 'elements' or some other non-committal term to avoid unnecessary controversy. Our feeling is that the elements are related in some direct way to low level structures in the brain, with the exact details currently unclear.

For reviews of such models see a book edited by Hinton and Anderson (1981) and two books by Kohonen (1977, 1984). A recent paper by McClelland and Rumelhart (1985) discusses at length

distributed memories and representation of knowledge from an approach similar to ours. They present a number of computer simulations of systems related and complementary to the simulations we present here.

The Linear Associator. Abstractly, a synapse connects two neurons together, so the activity of the presynaptic neuron affects the activity of the post-synaptic neuron. Most neuroscientists believe that changes in the strengths of synapses are the physical locus of memory. It is easy to show that a 'generalized' synapse (i.e., connection between elements) of the kind first suggested by Donald Hebb in 1949. and called a 'Hebb synapse', realizes an associative system. Suppose that we have two sets of neurons, one projecting to the other, connected by a matrix of synaptic weights A , and that we wish to associate two activity patterns (state vectors) f and g . We assume A is composed of a set of modifiable 'synapses' or connection strengths.

We make two quantitative assumptions. First, the neuron acts to a first approximation like a linear summer of its inputs. That is, the i th neuron in the second set of neurons will display activity $g(i)$ when a pattern f is presented to the first set of neurons according to the rule,

$$g(i) = \sum_j A(i,j) f(j).$$

where $A(i,j)$ are the connections between the i th neuron in the second set of neurons and the j th neuron in the first set. We can write g as the simple matrix multiplication

$$g = A f.$$

Our second fundamental assumption involves the construction of the matrix A , with elements $A(i,j)$. We assume that these matrix elements (connectivities) are modifiable according to the generalized Hebb rule, that is, the change in an element of A , $\delta A(i,j)$, is given by

$$\delta A(i,j) \propto f(j) g(i).$$

Suppose initially A is all zeros, that is the system knows nothing. If we have a column input state vector f , and response vector g , we can write the matrix A as

$$A = \eta g f^T$$

where η is a learning constant. Suppose after A is formed, vector f is input to the system. A pattern g' will be generated as the output to the system according to the simple matrix multiplication rule discussed before. This output, g' , can be computed as

$$g' = A f, \\ \propto g.$$

We have generated a vector in the same direction as g ; the system has learned an association.

Linear Concept Formation Let us make the reasonable but powerful coding assumption that the activity patterns representing similar stimuli are themselves similar, that is, their state vectors are correlated. This means the inner product between two similar patterns is large. Now consider the case described above where the model has made the association $f \rightarrow g$. With an input pattern f' then

$$(\text{output pattern}) = g [f, f']$$

If f and f' are not similar, their inner product $[f, f']$ is small.

If f is similar to f' then the inner product will be large. The model responds to input patterns based on similarity to f . This rule is therefore related to the theory that categorization is based on similarity to a prototype. Knapp and Anderson, (1984) presented an application of this simple approach to psychological concept formation, specifically the learning of 'concepts' based on patterns of random dots.

Consider a situation in which a category contains many similar items. Here, a set of similar activity patterns (representing the category members) becomes associated with the same response, for example, the category name. It is convenient to discuss such a set of vectors with respect to their mean. Let us assume the mean is taken over all potential members of the category. Specifically consider a set of correlated vectors, $\{f\}$, with mean p . Each individual vector in the set can be written as the sum of the mean vector and an additional noise vector, d , representing the deviation from the mean, that is,

$$f_i = p + d_i$$

If there are n different patterns learned and all are associated with the same response, the final connectivity matrix will be

$$\begin{aligned} A &= \sum_{i=1}^n g f_i^T \\ &= n g p^T + \sum_{i=1}^n d_i^T \end{aligned}$$

Suppose that the term containing the sum of the noise vector is relatively small, as could happen if the system learned many randomly chosen members of the category (so the d 's cancel on the average and their sum is small) and/or if d is not large. In that case, the connectivity matrix is approximated by

$$A = n g p^T.$$

The system behaves as if it had repeatedly learned only one pattern, p , essentially the n times the mean of the set of vectors it was exposed to. Under these conditions, the simple association model extracts a the prototype just like a signal averaging program. In this respect the distributed memory model behaves like a psychological 'prototype' model, because the most powerful response will be to the pattern p , which it may never have seen.

However if the sum of the d 's is not relatively small, as might happen if the system only sees a few patterns from the set and/or if d is large, the response of the model will depend on the similarities between the novel input and each of the learned patterns, that is, the system behaves like a psychological 'exemplar' model.

We can also begin to see how the system can use partial information to reason 'cooperatively'. Suppose we have a simple memory formed which has associated an input f_1 with two outputs, g_1 and g_2 , and an input f_2 with two outputs g_2 and g_3 so that

$$A f_1 = g_1 + g_2 \quad \text{and}$$

$$A f_2 = g_2 + g_3.$$

Suppose we then present f_1 and f_2 together. Then, we have

$$A(f_1 + f_2) = g_1 + 2g_2 + g_3,$$

with the largest weight for the common association. This

consequence of superposition has let us pick out the common association of f_1 and f_2 , if we can suppress the spurious responses.

The cooperative effects described in several contexts above depend critically on the linearity of the memory since things 'add up' in memory. We will demonstrate below, in Simulation One, that it is easy to remove the extra responses due to superposition and generate the exact, noise-free output response with a non-linear categorization algorithm.

Dot Prototypes. Knapp and Anderson (1984) applied this model to the formation of simple 'concepts' composed of nine randomly placed dots. These stimuli were used in a classic experiment on concept formation by Posner and Keele (1968, 1970). Many variants of the initial experiments have been performed, with many different kinds of stimuli, but the general pattern of results is consistent. For a complete literature review, see Knapp and Anderson (1984) and McClelland and Rumelhart (1985).

The generic prototype experiment is quite straightforward. Described in terms of the Posner & Keele stimulus set, the experimenter starts with the generation of several (usually three or four) patterns of dots, randomly placed on a computer screen. These initial patterns are called 'prototypes'. Then distortions of the prototypes are generated by the experimenter, for example moving the individual dots in a random direction by a given amount. In the jargon, these distortions are called 'exemplars.' If the distance moved is small, the exemplars are visually similar to the prototypes; if the distance moved is large, an exemplar may appear to be a completely new pattern.

The experiment is divided into two parts, a learning phase and a testing phase. During the learning phase, subjects are presented with exemplars. They are taught (usually with feedback after a response) to group all the exemplars derived from a particular prototype with the same response. Subjects are required to reach a criterion number of correct responses or are given a fixed number of learning trials. During the testing phase, subjects are presented with stimuli and told to decide which category a stimulus belongs in. There are three kinds of test stimuli: first, the old exemplars, second, new exemplars generated by the same rules as the old ones, and, third, the prototype itself.

Experimental results vary, depending on details of the stimuli and task manipulations. However, in many cases and for many kinds of stimuli, experimental results show that the fastest response, the fewest errors in classification, and the strongest subjective certainty of correctness are found when the prototype is presented, which, of course, was never actually seen during the learning phase. Sometimes there is an advantage for the old exemplars and sometimes old and new exemplars are apparently indistinguishable, depending on details of the experiment.

The model presented in Knapp and Anderson for these results is a straightforward realization of the linear model presented above. The coding of the state vectors was the key assumption for the model. It was assumed that there was a sensory surface with a topographic representation of visual space on it, inspired by the many such topographic maps in visual cortex. A dot in the world gives rise to a 'bump' of activity with exponential falloff on the

surface. Nine dots (the traditional experimental number) gave rise to a pattern on the surface of nine bumps, arranged as the stimulus was in the world.

Since all the exemplars derived from a given prototype are associated together, outer products of stimulus and response are stored in the memory matrix, A , during the learning phase. Since we know the codings of the stimuli, we therefore know part of the outer product. The responses are somewhat arbitrary, since we do not know how the responses are represented, but the selectivity of the system (what we are observing experimentally) is primarily determined by the inputs, as long as the output patterns are equivalent in terms of the model (i.e., equal length, equal separation between the responses, etc.)

Since this is a linear model, the output will generally be a superposition of the learned responses, but the biggest response will be that associated with the appropriate prototype. The amplitude of the associated response will be a measure of response time, certainty, etc., and that was used in fitting the data in the paper. In general this simple model gave a good account of itself in fitting a set of experimental data.

However, the associated response was never actually generated, merely a quantity related to its size, and the actual output was always, potentially, a sum of responses and not a single, clean, output. We claimed earlier that we could develop a system to 'clean up' an output with noise, or an output which was a superposition of responses, and below we present a simple system to do this, which is, of necessity, non-linear. In the section below entitled, Linear Prototypes, Revisited, we show the

operation of such a system applied to a prototype experiment.

Error Correction and the BSB Model. For many applications the linear model is too noisy. Given a learned association $f \rightarrow g$, and many other associations learned in the same matrix, the pattern generated when f is presented to the system may not be close enough to g to be satisfactory. By using an error correcting technique related to the Widrow-Hoff procedure we can force the system to give us correct associations. (This technique is referred to as the 'delta' method in McClelland and Rumelhart, 1985, because it is learning the difference between desired and actual output.) Suppose information is represented by vectors associated by $f_1 \rightarrow g_1, f_2 \rightarrow g_2 \dots$. We wish to form a matrix A of connections between elements to accurately reconstruct the association. First, a vector, f , is selected at random. Then the matrix, A , is incremented according to the rule

$$\Delta A = \eta (g - Af) f^T$$

where ΔA is the change in the matrix A and where the learning coefficient, η , is chosen so as to maintain stability. The learning coefficient can either be 'tapered' so as to approach zero when many vectors are learned, or it can be constant, which builds in an interesting 'short term memory' because recent events will be recalled more accurately than past events. As long as the number of vectors is small, this procedure is fast and converges in the sense that after a period of learning,

$$Af = g.$$

If $f = g$, the association of a vector with itself is referred to by Kohonen as an 'autoassociative' system. Suppose we are interested in looking at autoassociative systems,

$$\Delta A = \eta f_1 f_1^T$$

where η is some constant.

Let us consider the simple case where only one state vector is stored. We can then see how feedback can reconstruct a missing part of an input state vector. To show this, suppose we have a normalized state vector f , which is composed of two parts, say f' and f'' , i.e. $f = f' + f''$. Suppose f' and f'' are orthogonal. One way to accomplish this would be to have f' and f'' be subvectors that occupy different sets of elements -- say f' is non-zero only for elements $[1..n]$ and f'' is non-zero only for elements $[(n+1)..Dimensionality]$.

Then consider a matrix A storing only the single autoassociation of f that is

$$A = (f' + f'')(f' + f'')^T$$

(Let us take $\eta = 1$). The matrix is now formed. Suppose at some future time a truncated version of f , say f' is presented at the input to the system. The output is then given by

$$\begin{aligned} (\text{output}) &= A f' \\ &= (f' + f'') [f', f'']. \end{aligned}$$

The rest of the state vector has been reconstructed. Thus, the model is able to fill in missing information, perhaps as people are able to do when they identify objects based on partial information.

If we combine error correction with autoassociation, we want the system to behave after learning a set of stimuli $\{f\}$ as

$$A f_i = f_i$$

Therefore, one way to view the autoassociative system combined with error correction is that it is forcing the system to develop a particular set of eigenvectors. The system is then noisier and more complex, but the reconstructive abilities remain.

How could we use such a system? Assume that we want to get associated information that we currently do not have, or we want to make 'reasonable' generalizations about a new situation based on past experience. We must have some information to start with. The starting information is represented by a vector constructed according to rules used to form the original vectors, except missing information is represented by zeros. Intuitively, the memory, that is the other learned information, is represented in the cross connections between vector elements and the initial information is the key to get it out. The retrieval strategy will be to repeatedly pass the information through the matrix A and to reconstruct the missing information using the cross connections. Since the state vector may grow in size without bound, we limit the elements of the vector to some maximum and minimum value.

We will use the following nonlinear algorithm. Let $f(i)$ be the current state vector of the system. $f(0)$ is the initial vector. Then, let $f(i+1)$, the next state vector be given by

$$f(i+1) = \text{LIMIT} [\alpha A f(i) + \gamma f(i) + \delta f(0)] .$$

The first term $(\alpha A f(i))$ passes the current state through the matrix and adds more information reconstructed from cross connections. The second term $\gamma f(i)$ causes the current state to

decay slightly. This term has the qualitative effect of causing errors to eventually decay to zero as long as γ is less than 1. The third term, $\delta f(0)$ can keep the initial information constantly present. Sometimes δ is zero and sometimes δ is non-zero depending on the requirements of the task. If $\delta = 1$, this is sometimes referred to as 'clamping' in the Boltzmann machine literature.

Once the element values for $f(i+1)$ are calculated, the element values are 'limited'. This means that element values cannot be greater than an upper bound or lower than a lower bound. This process contains the state vector within a set of limits, and we have called this model the 'brain state in a box' or BSB model.

Because the system is in a positive feedback loop but is limited, eventually the system will become stable and will not change. This may occur when all the elements are saturated or when a few are still not saturated. This final state will be the output of the system. The final state can be interpreted according to the rules used to generate the stimuli. This state will contain the directed conclusions of the information system. It will have filled in missing information, or suggested information based on what it has learned in the past, using the cross connections represented in the matrix. The dynamics of this system are closely related to the 'power' method of eigenvector extraction.

We have showed in the past (Anderson, Silverstein, Ritz, and Jones, 1977) that in the simple case where the matrix is fully connected (symmetric by the learning rule in the autoassociative system) and has no decay, that the vector will monotonically

lengthen. We would like to point out that the dynamics of this system are nearly identical to those used by Hopfield (1984) for continuous valued systems. It is one member of the class of functions he discusses, and can be shown to be minimizing an energy function. In the more general autoassociative case, where the matrix is not symmetric because of limited connectivity (i.e., some elements are identically zero) and/or there is decay, the system can be shown computationally to be minimizing a quadratic energy function (Golden, 1985). In the simulations to be described, the Widrow-Hoff technique is used to 'learn the corners' of the system, thereby ensuring that the local energy 'minima' and the associated responses will coincide.

In the language most favored in the Los Alamos conference in May, 1985, such a system becomes 'a dynamical system with attractors'. The location of the attractors in state space can be controlled by the learning algorithm.

General Description of Simulations. In the specific examples of state vector generation that we will use for the simulations to follow, letters, words and sets of words are coded as concatenations of the bytes in their ASCII representation. A parity bit is present. Zeros are replaced with minus ones. For example, an 's', ASCII 115, is represented by -1 1 1 1 -1 -1 1 1 in the state vector. A 200 dimensional vector would represent 25 alphanumeric characters. This is a 'distributed' coding because a single letter or word is determined by a pattern of many elements. It is rather arbitrary but it gives useful demonstrations of the power of the overall approach and lets internal structure of a stimulus be seen clearly. In the outputs from the simulations the

Anderson & Murphy, Concepts

underline, '_', corresponds to all zeros (for the input state vector) or to an uninterpretable character whose amplitude is below an interpretation threshold (for later iterations). That is, the output strings displayed are only those characters of which the system is 'very sure' because their actual element values were all above a high threshold. The threshold is only for our convenience in interpreting outputs; the full values are used in the computations.

All the BSB simulations below are 200 dimensional (25 characters). The autoassociative matrix is usually 50% connected, that is, about half the elements, randomly chosen, are set identically to zero. This could be interpreted as one neuron not speaking to another because they are not connected. During learning, items to be associated are chosen randomly from the stimulus set. Generally, for the sizes of simulation described below about 20-30 presentations, on the average, of each f-g pair is adequate to give essentially perfect learning, that is the cosine of the angle between the actual and desired output vectors is greater than 0.99.

In all the applications of the BSB model described here, the inputs are 'clamped', that is, $\delta = 1$. Simulations were done both with inputs clamped and unclamped and there was no significant difference in results, except for small changes in number of iterations required for saturation.

Simulation One: Linear Prototypes, Revisited. The claim was made that we could use a simple non-linearity to make the final output state vector consist of a single response, rather than a superposition of several responses. The paper analyzing the dot

Anderson & Murphy, Concepts

pattern experiments (Knapp & Anderson, 1984) used only the linear model, and used what could be construed as a non-linear decision process to chose the best response.

With the BSB model, we can see if the non-linear system will work the same qualitative way as the linear system, i.e. whether it will show prototype effects, even though only a single response is generated as the output. In the simulations to be described, responses were virtually always correct. To make contact with the data, it is necessary to find some kind of parameter that corresponds to a quantity that can be measured experimentally in human subjects.

The most obvious parameter is what might be called 'reaction time'. It is the number of matrix iterations required to meet some criterion: all elements at a limit, say, or all elements above an interpretation threshold. This is a measure of processing time. There is a huge psychological literature on reaction times, with many careful studies and a number of consistent results. We have used reaction time in both the linear and non linear models as a response measure in previous applications of the neural model to experimental data (Anderson, 1973; Anderson et. al., 1977). It would be expecting too much to ask for immediate quantitative agreement with data, but we should expect to see a number of qualitative effects that we can check fairly easily. Examples would be the consistent tradeoff between speed and accuracy, and the general result that the surer the system is of a response (i.e., if the response is consistent with a great deal of stored information) the faster is the response. (Anderson, in press)

We should expect that the prototype should therefore show the fastest 'reaction time' in the simulations, as well as in the experimental data. In the dot pattern experiments this is found, and is a general qualitative feature of the experimental literature.

Rather than using dot patterns, which would have required forbidding amounts of computer time, we used a small, 200 dimensional system using alphanumeric characters. Table 1 gives the stimulus sets that were learned and Table 2 gives the test stimuli used to test the resulting matrices. The rules for construction of the stimuli are straightforward: there are four 'prototypes': lists of five mammals, fish, grains, or trees. Exemplars are constructed by replacing the names of two of the members of the list with dashes, '-', that is, no learned exemplar contained all the information in the prototype. Replacing one character with another one ensured the vectors were always of constant length. New exemplars were constructed by replacing different sets of characters by dashes. The prototypes, the unmodified lists, were not presented during the learning phase.

Table 1 about here

Table 2 about here

This system used two matrices: specifically, a matrix was constructed using the linear system to associate the inputs (the f 's) and the outputs (the g 's). The outputs (the g 's) were then learned by an autoassociative system using Widrow-Hoff error correction. A test vector, say t_1 , was presented and an output vector, say o_1 was generated by

$$o_1 = A t_1$$

The output then served as the input to the non-linear BSB model.

Number of iterations required for the output vector to fully limit are plotted for various test stimuli in Table 3. It can be seen that the results of the simulations are a good replication of the basic pattern of experimental results found in a number of experiments. Prototypes are responded to fastest, though they were never presented during learning. There was also a slight advantage for old examples. The system made rare mistakes. The simulation presented here made none. Reaction times to mistakes were long, over 100 iterations. Clearly the same effects are demonstrable in the nonlinear model as in the linear one, with the attractive feature that this is now a general classification algorithm (it actually constructs the correct response) with some psychological support.

Table 3 about here

In a large simulation, using this technique, we claim we could have reconstructed the pattern of results in the Posner-Keele experiments from the initial stimulus coding to the firing pattern of the motor output. The attractive feature of

doing the simulation this way would be that in a simple learning system we would have driven some theoretical muscles to push the appropriate pseudo-button and as a side effect have reproduced the observed pattern of experimental results.

Simulation Two: Common Associations. A major theoretical problem in psychology, and a major practical problem in Artificial Intelligence, one particularly pronounced in language behavior, is disambiguating ambiguous stimuli. Alan Kawamoto's recent Ph.D. thesis (1985) discussed this problem, and presented a system which was capable of performing simple disambiguation of word like stimuli using a number of techniques including adaptation. Adaptation was also applied to a number of classic 'multi-stable' stimuli such as the Necker cube with some qualitative agreement with experimental data. (Kawamoto and Anderson, 1985). The adapting system worked, but was fairly sensitive to parameters. Although adaptation is undoubtedly necessary in a general system, it is possible to do robust disambiguation in some situations without using adaptation. An example of this important effect is demonstrated in Simulation Two. The input stimulus set is given in Table 4.

Table 4 about here

The particular problem is one that was suggested as a key problem of Artificial Intelligence by Eugene Charniak. If one says to a resident of North America, 'bat', 'ball', and 'diamond', everyone knows quickly that we are talking about baseball. (This has been tested in many lectures with the expected results.) The

Anderson & Murphy, Concepts

baseball association is only one possible association of each of these ambiguous words, yet we were able to pick out the common association quickly. If we give more information (i.e., it is a game) reaction time drops. Hints help. Yet, if we were doing some kind of combinatorial tree search, the more information we have, the more possible permutations of association combinations would have to be explored. In this case hints probably help for increased accuracy, but certainly must hurt reaction time.

We set up a simple simulation to illustrate how effective the distributed models are for disambiguation. The stimuli are autoassociated using the Widrow-Hoff procedure in a matrix with 50% connectivity. Note that in the nine stimuli that 'Game', 'Bat', 'Ball' and 'Diamd' are triply redundant: that is, each is used by three different possible stimuli. If a test stimulus such as 'Ball' is used, there are three possible final states containing 'Ball', each equally 'correct' since no more information was given. Which one is actually chosen by the simulation is somewhat random, and depends on the short-term history of the learning and on the structure of the particular connections used.

In two runs of Simulation Two, when 'Bat' was the test input, the output was 'Vampire' (f[2]), 'Ball' gave rise to 'Tennis' (f[5]) and 'Diamd' gave rise to 'GeoShape', f[7]. (The other ten simulations that are not described here had one or the other test stimulus give rise to 'Baseball', which did not make the desired point.) The average number of iterations for complete saturation of the single inputs was 98.5 iterations. The input and output vectors for various conditions are given in Table 5.

Table 5 about here

If a test stimulus composed of two items, i.e. 'Bat' and 'Ball' or 'Bat' and 'Myth' the appropriate answer was chosen, 'Baseball' or 'Vampire' respectively, even though each word separately gave a different output. The average number of iterations for such an input was 38.33. If triples were used, the average number of iterations was 20.25 and if all four ambiguous items were used as the input, only 14 iterations were required. Clearly, in this case additional information helped and greatly decreased 'reaction time.'

The point of this simulation is that ambiguity is no problem for this system and disambiguation by context takes place in a rapid and natural way. A 'concept' such as baseball can be formed from pieces with multiple meanings and with predictable reaction time patterns for information retrieval.

Simulation Three: Learning a Concept Hierarchy. Up to this point, we have been teaching the model concepts at one level of abstraction. However, as discussed earlier, people do not only divide the world up this way--they also use concepts at different levels of abstraction, from very specific to very general. Often, these concepts are arranged hierarchically.

As a first attempt to see if the model will learn hierarchically organized concepts, we taught it the set of stimuli shown in Table 6. These items could be divided into tools and vehicles at the highest level, but these two groups were further

divided into more specific categories: saw, hammer, screwdriver, drill, wrench, pliers, car, plane, train. Finally, each of these consisted of one or more specific examples, e.g., the hammer consisted of a claw hammer, rubber hammer, and ball-peen hammer. The main question we had about this set of stimuli was whether the model could learn it at all. Each item could be classified in a number of ways, and it was unclear whether the model would be able to deal with this ambiguity.

 Table 6 about here

The hierarchical organization was represented in two different ways. First, the fact that all the cars had the string 'Car ' in them was the primary clue that they were all in the same class. Second, the more general categories were represented as separate items. For example, the item: ' DrillShopToolMaksThngs' represents the fact that drills are tools and that they're used in the shop to make things (the spaces at the beginning of the item indicate that this is a "generic" drill rather than a specific one, like a Drill Press). Would the model be able to handle both types of information without becoming confused?

The details of the simulation were similar to those described previously. This is a 200 dimensional, 50% connected autoassociative system. The system learned about 1500 total random presentations, about 50 presentations per item. The system learned accurately. During retrieval, inputs were clamped, i.e. the initial stimulus was added to the state vector after every

iteration.

After the learning phase, we tested the categorization ability of the model with a number of different items. Overall, we found that it was able to make both specific and generic inferences at appropriate times, and that it was able to recover items using incomplete information. First, when given a category name (e.g., 'Hamr', 'ScrD', 'Plne'), it produced information about items in that category. Table 7 gives a several examples of response to simple probes. For example, it was able to say that hammers are held in the hand, hit things, have a head and a handle and are used to hit metal. The first three attributes are true of all hammers but the last is true of two out of the three hammer exemplars. This behavior, outputting the most frequent attribute, is typical. When asked about cars, it found that cars have wheels, there are four of them (both true of all cars) and also that they ride on the street and are common (true of 2/3 cars). Thus, the model is able to extract the typical features of the category in this situation. Often the system could not give a specific example of a hammer or saw, say, but would instead give no interpretable or nonsense characters in the first four positions in the state vector. In cases where only one or two examples of a category had been presented (screwdrivers or wrenches) it did produce a specific name. (See the section, Retrieval Techniques, below for further discussion of this point.) The model will abstract a general concept about many similar items but will remember a small set of dissimilar items as 'specific examples', just as we would predict from the prototype experiments, discussed earlier.

Table 7 about here

A second test of the model is to give it a feature and see if it can recover the appropriate category. It is often better at this task, because the features are more distinctive than the category names. When given the feature "Rails," the model recovers the information that the item is a train with many wheels, run by Amtrak with a Diesel engine.

Some temporal patterns for retrieval seem to emerge from the data. The data in Table 7 are arranged so that the first correctly interpreted appearance of each feature is listed. So, when 'Saw' is the input, then the specific saw features (Teeth and Cuts) emerge rather early and the name of a specific saw type never occurs. When 'Teeth' is an input, 'Cuts' appears quite quickly, followed after a long interval by other information. When both 'Saw' and 'Teeth' are input together, 'Cuts' follows after only 10 iterations. Note that all simple tools inputs have 'Hand' emerge more quickly than any other feature. This seems to be because 80% of the state vectors at the level of specific examples contain the string 'Hand.' However, the system could retrieve power tools (e.g., 'BandSaw ') if that was specifically probed. Note that 'Powr' is the last feature retrieved for 'BandSaw '.

The simulations show a number of errors. These errors were revealing of the conceptual structure created by the model. For example, the model did not learn the Drill category well. When presented with the name 'Dril', it generated the incorrect

features 'Cuts', 'Teeth', and 'Metl.' The first two features are properties of the Saw category and the last is a property of two hammers and a saw exemplar. One might have expected that any errors would be simply random: if the model didn't learn the category, then it should generate features nonsystematically. Clearly, this did not happen. Instead, the model classified the drills with the other tools; when it becomes confused about the answer, it responds with features from the same superordinate category. Furthermore, the error shows a typical pattern similar to the 'family resemblance' structure of human concepts (Rosch & Mervis, 1975), in that the most frequent features are the ones that the model outputs. When it is unsure about a tool, it produces features like 'Cuts' and 'Metl' that are found in many tools. The simulations produced a number of examples like this one--all in the tool category, presumably because it was larger and its items overlapped more than the vehicle category. We discuss error correction techniques in the next section.

Finally, we should ask how the model did in learning the explicit superordinate categories, tool and vehicle. When the model was presented with "generic items" (those with the spaces in front of the category name--see example 8 in Table 7), it correctly chose generic information, that the object was a tool and made things or that it was a vehicle used for transportation. Furthermore, the model would recover generic information when presented with generic features. So, when told that an object was used in the shop, it could identify it as a tool used to make things.

Perhaps these abilities are not surprising. After all, the model was explicitly taught the exemplars and generic concepts, so it could well be expected to be able to recover them when given partial information. But a more difficult task was to provide the model with a mixture of generic and specific information, for example, to tell it that something was used in the shop (a generic tool feature) and had jaws (a feature of the pliers exemplar). In this case, the model was able to identify the object as a pliers and add the features of being used to hold things. These cases are important in showing that the specific and generic information are not being learned as unrelated facts. Clearly, the model "knows" that the information about pliers in general is related to the information about particular kinds of pliers. In short, the model has represented a hierarchically structured set of information, even though it does not use the network formalism common to most hierarchical models of knowledge representation (e.g., Collins & Quillian, 1969; Fahlman, 1979). Because the generic information about the vehicles and tools consists of different features than are stored in the specific items (following the principle of cognitive economy), it is possible for the model to keep the two kinds of information separate. Yet, because they involve the same concept names, it is also possible for the model to connect them up when necessary.

Although this stimulus set is idealized in a number of ways, it is a first step towards developing a memory model that can represent what Rosch (1978) called the "horizontal" and "vertical" dimensions of concepts at the same time. That is, the model does well in abstracting the properties of all saws or all cars, in spite of the differences among the individual items. And the

model also does well in representing the relations between concepts at different levels of abstraction: ball-peen hammer to hammer to tool. In future work, we hope to extend the model so that it will discover the hierarchical relations on its own. In the case presented here, we have told it the properties of tools and vehicles and quite explicitly grouped together the members of each superordinate category (through the "generic" items). However, people can discover the relations between categories at different levels of abstraction (Murphy and Smith, 1982), and it is our goal to show that the model can do it as well, given the proper learning set.

Retrieval Techniques. A significant number of errors were made in Simulation 3, because many of the stimuli were very similar and because of their complex internal structure. Sometimes part of the state vector could not be reconstructed at all, after hundreds of iterations. For example, the the erroneous or incomplete outputs given in Table 8 did not change no matter how many additonal iterations were used. They had obviously hit an undesirable equilibrium. In terms of the Boltzmann model literature, we might say that they were trapped in a local minimum of the energy function which was not the desired global minimum.

One way to get out of the local minimum, one used effectively in the Boltzmann approach, is deliberately to add noise to the system. We have used this technique in the past and it does not seem to work well in this system. Small amounts of noise do not work at all, and large amounts introduce uncontrollable errors as well as operate very slowly. Noise is also foreign to our deterministic approach.

Are there techniques we can use that will significantly increase our chance of getting correct answers? In particular, are there techniques that can be applied mechanically and which will operate quickly if an error is detected? All but one of the stimuli learned in Simulation Three contain no zeros. Desirable final output states will be fully saturated. Therefore, one practical way of detecting errors is to assume an error exists if a stable state is generated which is not fully saturated. Another way would be to assume a simple dictionary containing allowable sub-groupings of letters -- 'words'. If a stable nonsense word appears, then an error exists. For example, 'Fasn' gives rise to the stable state 'StztScrDHandFasnPtipScrew', and the first four letters are not an allowable string.

Given an error that the system has detected, how can it be rectified? There are several simple techniques that appear to be quite effective. Consider the rather large number of obvious errors seen in the outputs listed in Table 7. Suppose when the system gets stuck -- i.e., in a stable state that contains an error, we reset to zero the element values of the word that has the problem. Notice that the errors are almost always confined to at most one or two 'words'. Table 8 shows the use of this technique to correct most of the errors in Table 7 as well as some additional tests. A few errors remain: there are two confusions of 'Saw' and 'Dril' which cannot be removed. Apparently the definitions of these two tools are sufficiently confused by the system that it never gets them fully correct.

Table 8 about here

Anderson & Murphy, Concepts

The use of short-term decay to correct minor errors also can be seen in Part 3 of Table 8, where upper case letters were used in the input in place of the correct lower case letters. When the input was not clamped (i.e., $\delta = 0$ in the algorithm) the system corrected the mistake. When the input was clamped, it could not change. There was, in general, no obvious advantage for clamping, though 'reaction times' were usually slightly less in the clamped system.

There is scope for a good deal of creativity in the retrieval process that does not necessarily involve the use of deliberately added noise. The advantage of simple deterministic techniques such as the ones used here are that they are fast, can be made automatic, and seem to be straightforward and predictable in operation. A collection of effective retrieval techniques can significantly add to the power of distributed models.

References

Anderson, J.A. (1970). Two models for memory organization using interacting traces. Mathematical Biosciences., 8, 137-160.

Anderson, J.A. (1973). A theory for the recognition of items from short memorized lists. Psychological Review., 80, 417-438.

Anderson, J.A. (in press). In E. Bienenstock, F. Fogelmann, and G. Weisbuch (Eds.), Cognitive capabilities of a parallel system. Disordered Systems and Biological Organization. Berlin: Springer.

Anderson, J.A., Silverstein, J.W., Ritz, S.A. & Jones, R.S. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. Psychological Review. 84, 413-451.

Cohen, B., & Murphy, G.L. (1984). Models of concepts. Cognitive Science., 8, 27-58.

Collins, A. & Quillian, M. (1969). Retrieval times from semantic memory. Journal of Verbal Learning and Verbal Behavior. 8, 241-248.

Fahlman, S.E. (1979) NETL: A System for Representing and Using Real-World Knowledge. Cambridge, MA: MIT Press.

Golden, R. (1985) Identification of the BSB neural model as a gradient descent technique that minimizes a quadratic cost function over a set of linear inequalities. (Submitted for publication).

Hinton, G.E. & Anderson, J.A. (Eds.), Parallel Models of Associative Memory. Hillsdale, N.J.: Erlbaum Associates.

Hinton, G.E. & Sejnowski, T.J. (1984). Optimal pattern inference. IEEE Conference on Computers in Vision and Pattern Recognition.

Hopfield, J.J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Natl. Acad. Sci. U.S.A., 81, 3088-3092.

Jackendoff, R. (1983). Semantics and Cognition. Cambridge, MA: MIT Press.

Kawamoto, A. (1985). Dynamic Processes in the (Re)Solution of Lexical Ambiguity. Ph.D. Thesis, Department of Psychology, Brown University.

Kawamoto, A. & Anderson, J.A. (1985). A neural network model of multistable perception. Acta Psychologica., 59, 1-31.

Knapp, A.G. & Anderson, J.A. (1984). Theory of categorization based on distributed memory storage. Journal of Experimental Psychology: Learning, Memory, and Cognition., 10, 616-637.

Kohonen, T. (1977). Associative Memory. Berlin: Springer.

Kohonen, T. (1984). Self Organization and Associative Memory. Berlin: Springer.

McClelland, J.L. & Rumelhart, D.E. (1985). Distributed memory and the representation of general and specific memory. Journal of Experimental: General. 114, 159-188.

Murphy, G.L. & Smith, E.E. (1982). Basic-level superiority in picture categorization. Journal of Verbal Learning and Verbal Behavior. 21, 1-20.

Posner, M.I. & Keele, S.W. (1968). On the genesis of abstract ideas. Journal of Experimental Psychology. 77, 353-363.

Posner, M.I. & Keele, S.W. (1970). Retention of abstract ideas. Journal of Experimental Psychology. 83, 304-308.

Rips, L.J., Shoben, & Smith, E.E. (1973). Semantic distance and the verification of semantic relations. Journal of Verbal Learning and Verbal Behavior. 12, 1-20.

Rosch, E. (1975). Cognitive representations of semantic categories. Journal of Experimental Psychology: General., 104, 192-233.

Rosch, E. (1978). Principles of categorization. In E. Rosch & B.B. Lloyd (Eds.), Cognition and Categorization., Hillsdale, NJ: Erlbaum.

Rosch, E. & Mervis, C.B. (1975). Family resemblances: Studies in the internal structure of categories. Cognitive Psychology., 7, 573-605.

Smith, E. E. (1978). Theories of semantic memory. In W.K. Estes (Ed.) Handbook of Learning and Cognitive Processes (Vol. 6)., Hillsdale, NJ: Erlbaum.

Anderson & Murphy, Concepts

Smith, E.E. & Medin, D.L. (1981). Categories and Concepts.
Cambridge, MA: Harvard University Press.

Table 1
Simulation One
Prototype Effects in the BSB Model
Learned Associations

Input State Vector	Output State Vector
FC 1]. -----BearsDogs Cats	GC 1]. LivingAnimalMammalHunter
FC 2]. Lions-----Dogs Cats	GC 2]. LivingAnimalMammalHunter
FC 3]. LionsTiger-----Cats	GC 3]. LivingAnimalMammalHunter
FC 4]. LionsTigerBears-----	GC 4]. LivingAnimalMammalHunter
FC 5]. -----TigerBearsDogs ----	GC 5]. LivingAnimalMammalHunter
FC 6]. -----PerchCbassTuna	GC 6]. LivingAnimalFishesEdible
FC 7]. Trout-----CbassTuna	GC 7]. LivingAnimalFishesEdible
FC 8]. TroutSalmn-----Tuna	GC 8]. LivingAnimalFishesEdible
FC 9]. TroutSalmnPerch-----	GC 9]. LivingAnimalFishesEdible
FC10]. -----SalmnPerchCbass-----	GC10]. LivingAnimalFishesEdible
FC11]. -----BarlyOats Rice	GC11]. LivingPlantsGrainsEdible
FC12]. Wheat-----Oats Rice	GC12]. LivingPlantsGrainsEdible
FC13]. WheatRye -----Rice	GC13]. LivingPlantsGrainsEdible
FC14]. WhearRye Barly-----	GC14]. LivingPlantsGrainsEdible
FC15]. -----Rye BarlyOats ----	GC15]. LivingPlantsGrainsEdible
FC16]. -----PoplrCheryApple	GC16]. LivingPlantsTrees Boards
FC17]. Oak -----CheryApple	GC17]. LivingPlantsTrees Boards
FC18]. Oak Pine -----Apple	GC18]. LivingPlantsTrees Boards
FC19]. Oak Pine Poplr-----	GC19]. LivingPlantsTrees Boards
FC20]. -----Pine PoplrChery-----	GC20]. LivingPlantsTrees Boards

Table 2

Simulation One

Test Stimuli Used in Prototype Simulation

Old Examples

TC 11]. -----BearsDogs Cats	TC111]. -----BarlyOats Rice
TC 21]. Lions-----Dogs Cats	TC121]. Wheat-----Oats Rice
TC 31]. LionsTiger-----Cats	TC131]. WheatRye -----Rice
TC 41]. LionsTigerBears-----	TC141]. WhearRye Barly-----
TC 51]. -----TigerBearsDogs -----	TC151]. -----Rye BarlyOats -----
TC 61]. -----PerchCbassTuna	TC161]. -----PoplrCheryApple
TC 71]. Trout-----CbassTuna	TC171]. Oak -----CheryApple
TC 81]. TroutSalmn-----Tuna	TC181]. Oak Pine -----Apple
TC 91]. TroutSalmnPerch-----	TC191]. Oak Pine Poplr-----
TC101]. -----SalmnPerchCbass-----	TC201]. -----Pine PoplrChery-----

New Examples

TC211]. --ons--ger--ars--gs --ts	TC311]. --eat--e --rly--ts --ce
TC221]. L--nsT--erB--rsD--s C--s	TC321]. W--atR-- B--lyO--s R--e
TC231]. Li--sTi--rBe--sDo-- Ca--	TC331]. Wh--tRy-- Ba--yOa-- Ri--
TC241]. Lio--Tig--Bea--Dog--Cat--	TC341]. Whe--Rye--Bar--Oat--Ric--
TC251]. -ion--ige--ear--ogs--ats	TC351]. -hea--ye --arl--ats--ice-
TC261]. --out--lmn--rch--ass--na	TC361]. --k --ne --plr--ery--ple
TC271]. T--utS--mnP--chC--ssT--a	TC371]. O-- P--e P--lrC--ryA--le
TC281]. Tr--tSa--nPe--hCb--sTu--	TC381]. Oa-- Pi-- Po--rCh--yAp--e
TC291]. Tro--Sal--Per--Cba--Tun--	TC391]. Oak--Pin--Pop--Che--App--
TC301]. -rou--alm--erc--bas--una-	TC401]. -ak --ine--opl--her--ppl-

Prototypes

Mammals

TC411]. LionsTigerBearsDogs Cats

Fish

TC421]. TroutSalmnPerchCbassTuna

Grains

TC431]. WheatRye BarlyOats Rice

Trees

TC441]. Oak Pine PoplrCheryApple

Table 3

Simulation One

Summary of a Typical Simulation: Iterations to Saturation

1. Category: Mammals	Prototype:	30
	Old Examples:	48.0
	New Examples:	57.8
2. Category: Fish	Prototype:	24
	Old Examples:	34.8
	New Examples:	33.2
3. Category: Grains	Prototype:	23
	Old Examples:	27.8
	New Examples:	26.8
4. Category: Trees	Prototype:	23
	Old Examples:	41.6
	New Examples:	47.8
5. Overall Average	Prototype:	25.0
	Old Examples:	38.1
	New Examples:	41.4

The number gives the number of iteration required for full saturation, i.e. for every vector element to be at a positive or negative limit. Other measures, for example, number of iterations for all letters to be unambiguously interpreted, showed the same pattern.

Table 4
Simulation Two

Learned Associations

Input Stimuli	Output Stimuli
FC 1]. BaseballGameBat BallDiamd	GC 1]. BaseballGameBat BallDiamd
FC 2]. Vampire MythBat NiteDracu	GC 2]. Vampire MythBat NiteDracu
FC 3]. Animal LiveBat WingFlyng	GC 3]. Animal LiveBat WingFlyng
FC 4]. Poker GameBeerTablCards	GC 4]. Poker GameBeerTablCards
FC 5]. Tennis GameCortBallRackt	GC 5]. Tennis GameCortBallRackt
FC 6]. Dancing RichPrtyBallSocty	GC 6]. Dancing RichPrtyBallSocty
FC 7]. GeoShapeTwoDCrclSgreDiamd	GC 7]. GeoShapeTwoDCrclSgreDiamd
FC 8]. GeoModelTreDSphrBallTetra	GC 8]. GeoModelTreDSphrBallTetra
FC 9]. ExpJewelRichRubyOpalDiamd	GC 9]. ExpJewelRichRubyOpalDiamd

Note: This is an autoassociative system so the input and output are identical.

Table 5

Simulation Two

Test Inputs and Associated Output Vectors

1. Single Ambiguous Words

Input	Fully Limited Output	Number of Iterations
Bat	→ Vampire MythBat NiteDracu	81
Ball	→ Tennis GameCortBallRackt	105
Diamd	→ GeoShapeTwoDCrclSgreDiamd	134
Game	→ Poker GameBeerTablCards	68

2. Pairs of Ambiguous Words

Bat Ball	→ BaseballGameBat BallDiamd	30
Bat Diamd	→ BaseballGameBat BallDiamd	28
BallDiamd	→ BaseballGameBat BallDiamd	27
Game Ball	→ Tennis GameCortBallRackt	91
GameBat	→ BaseballGameBat BallDiamd	28
Geo Diamd	→ GeoShapeTwoDCrclSgreDiamd	22

3. Pairs of Words

Bat Nite	→ Vampire MythBat NiteDracu	23
Bat Wing	→ Animal LiveBat WingFlyng	25
Shape Diamd	→ GeoShapeTwoDCrclSgreDiamd	22

4. Triples of Ambiguous Words

Bat BallDiamd	→ BaseballGameBat BallDiamd	18
GameBat Diamd	→ BaseballGameBat BallDiamd	18
GameBat Ball	→ BaseballGameBat BallDiamd	25

5. Quadruple of Ambiguous Words

GameBat BallDiamd	→ BaseballGameBat BallDiamd	14
-------------------	-----------------------------	----

Table 6.

Simulation Three

Autoassociative Stimulus Set

Input Stimuli

FC 1]. Rip Saw PowrCutsWoodTeeth	FC17]. HamrHomeToolMaksThngs
FC 2]. BandSaw PowrCutsAll Teeth	FC18]. ScrDHomeToolMaksThngs
FC 3]. HandSaw HandCutsWoodTeeth	FC19]. DrilShopToolMaksThngs
FC 4]. CopgSaw HandCutsWoodTeeth	FC20]. DrilHomeToolMaksThngs
FC 5]. HackSaw HandCutsMetlTeeth	FC21]. WrenHomeToolMaksThngs
FC 6]. ClawHamrHandHitsWoodHdHnd	FC22]. PlirHomeToolMaksThngs
FC 7]. RubrHamrHandHitsMetlHdHnd	FC23]. SprtCar FourWhlsComnStret
FC 8]. BaPeHamrHandHitsMetlHdHnd	FC24]. FamlCar FourWhlsComnStret
FC 9]. PhlpScrDHandFasnXtipScrew	FC25]. ClscCar FourWhlsRareConcr
FC10]. StrtScrDHandFasnPtipScrew	FC26]. Jet PlneTwo WingFastAtmos
FC11]. PresDrilPowrHoleAll Bit	FC27]. PropPlneTwo WingSlowAtmos
FC12]. HandDrilHandHoleAll Bit	FC28]. DeslTrneManyWhlsAmtkRails
FC13]. ScktWrenHandFasnBoltSoHnd	FC29]. Car Own VehiclTranspt
FC14]. LnNsPlirHandHoldPiecJaws_	FC30]. PlneTcktVehiclTranspt
FC15]. Saw ShopToolMaksThngs	FC31]. TrneTcktVehiclTranspt
FC16]. Saw HomeToolMaksThngs	

Note: This is an autoassociative system so the input and output stimuli are equal.

Table 7
Simulation Three

Input Stimulus		Output at Various Stages	Nr of Iterations
1.	Saw	→ Saw Hand M T et _ab Saw HandCut Me Teeth _ab Saw HandCutsMe Teeth _ab Saw HandCutsMetlTeeth	43 66 67 96
2.	Hamr	→ _h' HamrHand M f _pHamrHandHit M d _pHamrHandHitsMe HdHnd B HamrHandHitsMetlHdHnd	39 44 52 86
3.	ScrD	→ _h ScrDHandFasnXtip rew _h pScrDHandFasnXtipScrew PhlpScrDHandFasnXtipScrew PhlpScrDHandFasnXtipScrew	40 44 62 86
4.	Dril	→ _cDrilHand Me i Pc_cDrilHandCut MetlT th Pc_cDrilHandCutsMetlTeeth	42 78 85
5.	Wren	→ _b pWrenHandF s l _j WrenHandFasnBgltSo d Sck WrenHandFasnBoltSoH d Sck WrenHandFasnBoltSoHnd ScktWrenHandFasnBoltSoHnd	48 86 108 109 123
6.	BandSaw	→ BandSaw Cuts l Te th BandSaw P CutsAll Te th BandSaw P CutsAll Teeth BandSaw PowrCutsAll Teeth	50 53 55 120
7.	Cuts	→ Cuts Teeth _i_gSa H CutsWoodTeeth _i_gSaw H CutsWoodTeeth	16 49 57
8.	Saw	→ Saw opToolMak Thn s Saw ShopToolMak Thngs Saw ShopToolMaksThngs	38 43 45
9.	Shop Jaws	→ ShopHold i cJaws _q ShopHoldPiecJaws LnNs ShopHoldPiecJaws LnNsPlirShopHoldPiecJaws	15 22 40 99
10.	Car	→ _Car O WhlsC _zdCar No WhlsCo nS ret _zdCar ourWhlsComnStret _zdCar FourWhlsComnStret	27 63 72 74
11.	Rails	→ T_q M Wh AmtkRails T n M nyWhlsAmtkRails _slTrneM nyWhlsAmtkRails _eslTrneManyWhlsAmtkRails DeslTrneManyWhlsAmtkRails	17 19 26 28 48

Caption for Table 7

Time evolution of the output is sketched in this table. Each time a 'word' is successfully reconstructed, the output state vector at that iteration is listed. The number refers to the number of iterations.

Table 8

Simulation Three

Retrieval Techniques

1. Successful Corrections

_____Teeth →	_k_ Saw PowrCutsWoodTeeth	150
0000Saw PowrCutsWoodTeeth →	Rip Saw PowrCutsWoodTeeth	30
_____Cuts_____ →	Ck_gSaw H__CutsWoodTeeth	150
0000Saw _____CutsWoodTeeth →	CopgSaw HandCutsWoodTeeth	51
_____Saw _____Teeth →	_ce_ Saw PowrCutsAll Teeth	150
0000Saw PowrCutsAll Teeth →	BandSaw PowrCutsAll Teeth	33
_____Hamr_____ →	B_b_HamrHandHitsMetlHdHnd	150
0000HamrHandHitsMetlHdHnd →	BaPeHamrHandHitsMetlHdHnd	41
_____Car_____ →	__zdCar FourWhlsComnStret	150
0000Car FourWhlsComnStret →	SprtCar FourWhlsComnStret	44
_____Fasn_____ →	StztScrDHandFasnPtipScrew	150
0000ScrDHandFasnPtipScrew →	StrtScrDHandFasnPtipScrew	50

2. Unsuccesful Corrections

_____Dril_____ →	_c_ DrilHandCutsMetlTeeth	150
0000DrilHandCutsMetlTeeth →	HackDrilHandCutsMetlTeeth	52
_____Bit →	P_e_ Saw H__HoleAll Bit	150
0000000000000000Hole0000Bit →	HandSaw HandHoleAll Bit	152

3. Correction of Minor Error by Unclamping Input

_____SAW_____ →	Ha_ Saw HandCutsMetlTeeth	150
0000Saw HandCutsMetlTeeth →	HackSaw HandCutsMetlTeeth	36

The first line gives the input test state vector and its final state after 150 iterations. These final states were stable, in that they did not change even after very large numbers of additional iterations. The elements in the locations that were incorrect were set equal to zero, indicated by a zero, '0', and iterations continued. The number refers to the number of additional iterations required for complete limiting.

END

DT/C

8-86